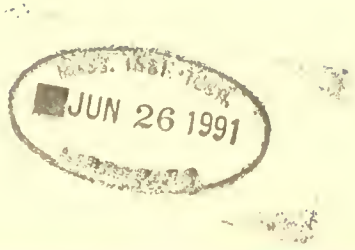
MIT LIBRARIES
3 9080 00686242 6

HD28
.M414
no.
3280-
91

DEWEY



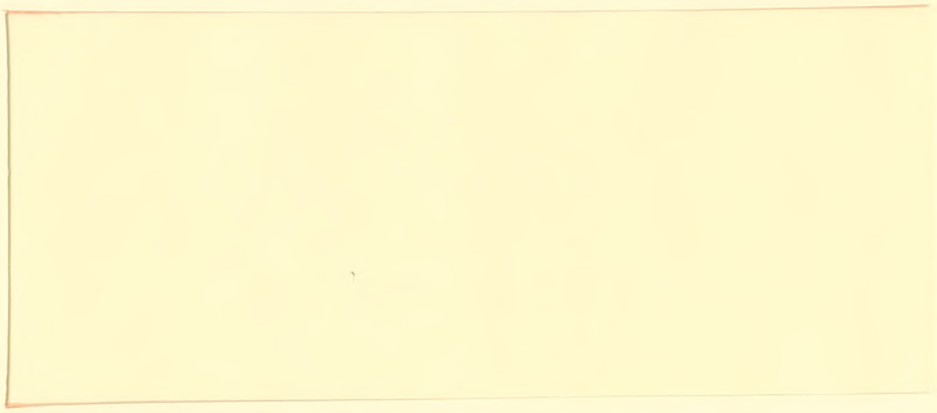
WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

Methods for Analyzing Design Procedures

David A. Gebala
Steven D. Eppinger

WP# 3280-91-MS April 1991

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139



Methods for Analyzing Design Procedures

David A. Gebala
Steven D. Eppinger

WP# 3280-91-MS April 1991

M.I.T. LIBRARIES
JUN 2 6 1991
RECEIVED

Methods for Analyzing Design Procedures

David A. Gebala
Steven D. Eppinger

Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

This paper surveys several common models which can be used to represent and study design procedures. A matrix representation of design is discussed in detail and both new and existing analysis tools are presented. Enhanced models are proposed to incorporate more information into the matrix representation and to allow more sophisticated analysis to be performed. The paper describes a testbed for the development and testing of algorithmic analysis tools and concludes with suggestions for further research into improving the design process.

1. Introduction

Product design is a complex activity requiring knowledge spanning many separate domains of study. Even the most rudimentary design activities demand scientific principles, engineering skills and often artistic creativity. Given the numerous inputs that feed into the design activity, it is not surprising that design presents a technical challenge even for relatively well understood products such as automobiles and airplanes. Because the vast amount of knowledge used in the design process is certainly more than any individual can manage, there is a need for clear representations which will aid our understanding of design procedures.

In addition to the tremendous magnitude of information involved, the complexity with which this information flows contributes significantly to the difficulty of design. For example, a path of information flow within a design activity may be circular. Information circuits are a consequence of design *iteration* - the revision of decisions which had been made using incomplete or imperfect information. Early decisions are checked and adjusted once complete information does become available, but an initial guess is necessary to initiate the iterative cycle of checking and adjusting.

In an effort to improve the execution of design procedures, the work reported here is aimed at [1] better representing the information requirements for the design activity, and [2] better understanding the structure of information flow in the design process. Our representation of the design process captures this information in a clear and concise manner which provides a global perspective on the information requirements at

each stage of the design activity. To improve the management of the design process, this paper will present algorithms which can be applied to strategically structure the design process and to suggest improved design sequences and tasks.

2. Design Process Modeling

The amount of information associated with the design process is enormous. Information is used, generated and transferred at each stage of the design task. Every decision is subject to constraints imposed by previous decisions, and in turn propagates new constraints to other decisions which are to be made at future stages of the design process. Keeping track of all such decisions and their consequences would be impossible for all but the most trivial design exercises. Attempting to document the complete design activity can be overwhelming.

The design of a complex industrial or consumer product can often involve hundreds or thousands of engineers and other professional over several years. This can quickly become an unwieldy activity which is difficult to manage. To make the design activity more tractable, managers find it convenient to follow a divide and conquer strategy. For example, one does not design an entire automobile without first breaking the task into functionally related sub-tasks. This design exercise might be decomposed into functionally structured areas such as chassis and suspension, drive train, brakes and wheels, body, etc. Within these areas one might find specialists in the applicable disciplines: structural, electrical, mechanical, etc.

The decomposition of the design activity into structured sub-units has been proposed by many authors as a first step to achieving a better understanding of the design process [1, 14, 16]. The goal is to decompose a large design activity into smaller, more tractable component design problems using the structure of the task as a guide. Through this decomposition, the design activity can be treated as a collection of smaller tasks. The information used and transferred by any one of these smaller tasks is considerably easier to track and manage. Following are several methods for representing and studying complex problems which are decomposed in this way. A simple design problem with seven tasks will be used as an example throughout.

2.1 Directed Graphs

Using a graphical mapping, one can represent the entire design activity as a system of interconnected nodes. The nodes can be chosen to represent individual sub-tasks, and the arcs linking the nodes can represent directed information flow. This system decomposition is easily depicted as a directed graph (digraph) [5]. An example of such a digraph is

presented in Figure 1. A link drawn from one node into another represents a required information transfer between the two nodes. In Figure 1, Task A requires information from Task D, but Task D does not require direct input from Task A.

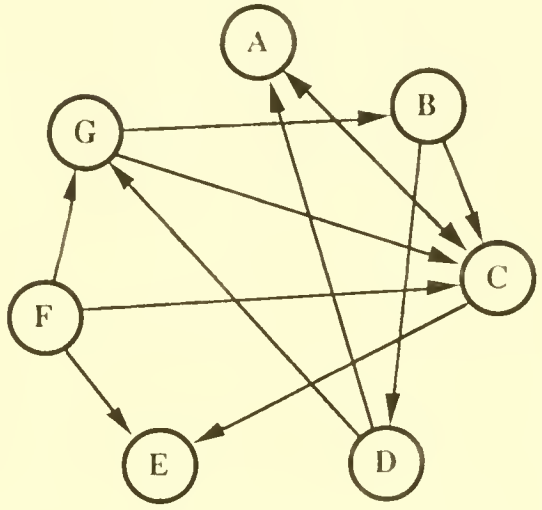


Figure 1. Directed Graph

Because the choice of node locations in the digraph is somewhat arbitrary, it is difficult to discern any structure in the digraph. In general, the digraph layout does not reflect any of the underlying structure of the design problem which it represents. Consequently, the digraph is useful when the number of nodes is small enough not to clutter the model and when the flow of information is not overly complex. As the number of nodes and links increases, the graph quickly becomes a cluttered and disorderly network of tangled arcs.

2.2 Program Evaluation and Review Technique

The digraph representation has been used as a basis for other models which aid in the management of multiple interacting units. If the nodes of a digraph are arranged along a time line, the result is similar to a Program Evaluation and Review Technique (PERT) chart. Usually the tasks are placed along the arcs, with nodes representing task completion milestones. The length of an arc is proportional to the activity's duration to provide a visual map of the project course. The time required to reach each node can be calculated to determine the critical path and to predict the expected completion date of the project [20]. Figure 2 depicts a typical PERT chart.

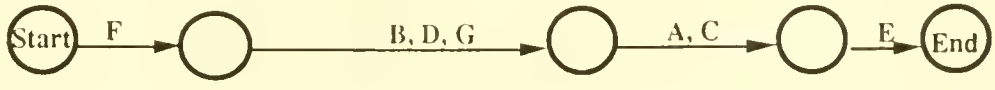


Figure 2. PERT chart

Although this model incorporates more information (such as timing) than the digraph on which it is based, it is still inadequate for representing the vast majority of design procedures where iteration is involved. The PERT chart is a natural representation for certain systems such as work flow on a shop floor or material movements in an assembly operation; however the shortcoming is that this project scheduling scheme cannot represent the the circular information flows often encountered in design. Loops are not allowed in a PERT chart and would be replaced with a single, composite activity which ignores the uncertainties associated with the loop. These uncertainties are the challenge in design management, and a useful representation must make these explicit.

2.3 Structured Analysis and Design Technique

One technique which has been widely used in documenting design procedures is the Structured Analysis and Design Technique developed by Ross [12]. This system of interconnected boxes and arrows is more formal than the digraph representation about depicting input and output, and attempts to overcome the size limitations by restricting the amount of information which can be placed on each page of the document. The result is a more orderly, structured model which does represent circuits, but which provides only limited glimpses into the design process. Unfortunately, loops remain indistinguishable from simple feed-forward transfer of information without explicitly tracing through multiple documents [10]. The SADT representation of the above digraph is presented in Figure 3. The digraph representation, the PERT chart, and the more structured SADT documentation suffer from practical size limitations and an inability to display circuits explicitly. Because they are cumbersome, we have found in practice that these methods are used primarily for documentation of design practices. These representations lack capacity to suggest improvements for effectively managing the design process.

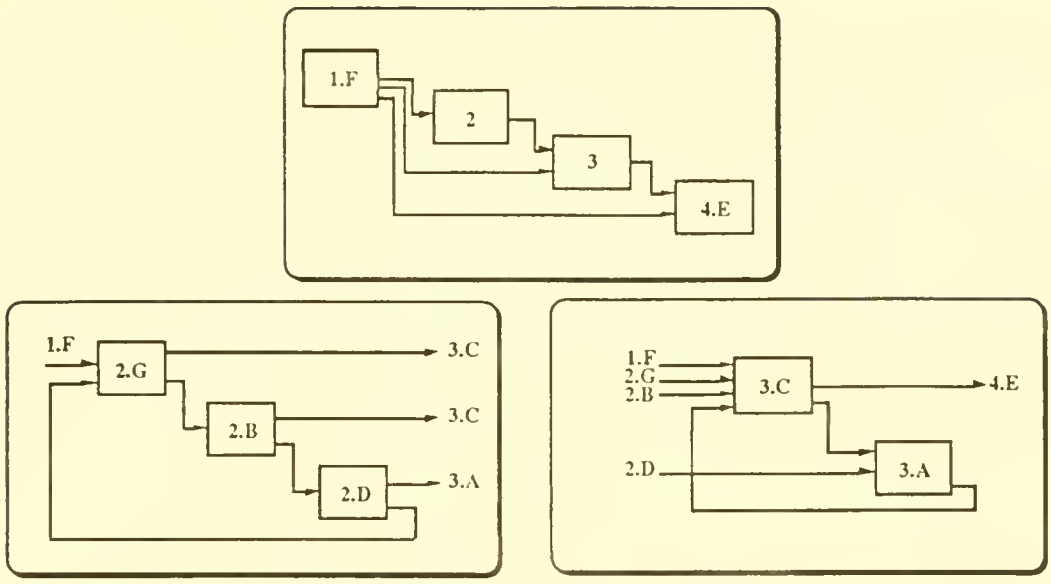


Figure 3. SADT Documents

2.4 Matrices

A representation which overcomes the size and complexity limitations of those discussed above is the design structure matrix [17]. This graphic representation uses the structure of the information flow to guide the decomposition of the design activity. The matrix embodies the structure of the underlying design activity by mapping the relations between tasks in a precise order which makes interdependence explicit.

The design structure matrix associated with a digraph is a square matrix which maps out the information links among individual design tasks. The matrix provides a systematic mapping that is clear and easy to read regardless of size. A design activity composed of n tasks would be represented as an $n \times n$ matrix. The task labels are placed down the side of the the matrix as row headings and across the top as column headings in the same order. The matrix element a_{ij} is non-zero if node i provides information to node j. The design structure matrix associated with the example presented above is shown in Figure 4. If one interprets the task ordering as a time sequence, the timing of information flow becomes explicit: marks below the diagonal represent information transferred to later tasks; marks above the diagonal depict information fed back to earlier tasks

The matrix representation has been used to model many different processes. Chemical engineers have used the matrix representation to model process flow sheets which contain recycle streams similar to the circuits described above [6]. Mathematicians have used this matrix form to solve simultaneous equations [9, 19]. Engineers have modeled the

interdependence of system parameters to determine the most efficient ordering of design decisions [2, 18].

Most recently, the matrix representation has been used as a management aid as well as an engineering tool to guide the organizational structure of design projects using the individual information requirements of the engineering tasks [4]. As an aid to better design management , we have used the matrix to identify tasks which are constrained to be serial by nature of the information flow connecting them. Conversely, the matrix also allows quick identification of tasks which can be performed in parallel. This representation has proven useful in studying the changes required in the implementation of concurrent engineering strategies [3].

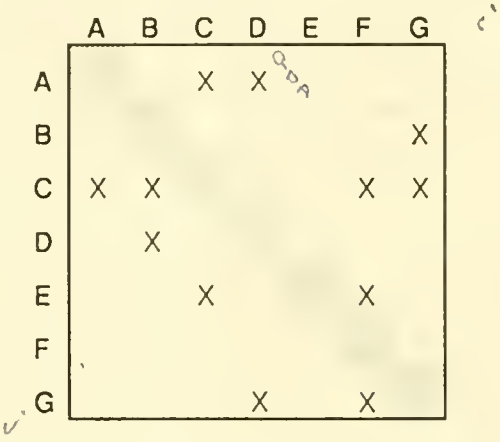


Figure 4. Design Structure Matrix

3. Design Process Analysis: Partitioning and Tearing

Once the design process has been mapped into a design structure matrix, the analysis proceeds in two separate stages. Borrowing the terminology proposed in the literature, we refer to the first stage of analysis as *partitioning* [13].

> *The goal of partitioning is to resequence the design tasks to maximize the the availability of information required at each stage of the design process.*

In some situations, the information flow will be such that not all the information can be made available when it is required. This indicates a circular path of information flow. A major outcome of the partitioning analysis is the identification of the tasks which are in these loops. Identification of loops is difficult to perform using a digraph, but can be considerably simplified using a design structure matrix. Partitioning identifies the tasks in a loop and clusters these tasks as a block on the diagonal of the design structure matrix.

When partitioning analysis identifies loops, each can be subjected to a second level of analysis. Borrowing terminology originated by Kron we refer to the detailed analysis within the loops as *tearing* [8]. Tearing is an analysis using both the sequence and interrelation of tasks within the blocks.

> *The goal of tearing is to resequence within the groups (blocks) of coupled tasks to find an initial ordering to start the iteration.*

To tear a relation implies the removal of a dependence or the assumption of some piece of information required to initialize iteration.

3.1 Partitioning Algorithm

The general strategy followed in partitioning is to systematically determine how to perform each task as early as possible in the design procedure. When all requisite information is available for a task, it is scheduled into the first space in the design structure matrix. When a task provides no information to future tasks, it is scheduled into the last space in the matrix. When no such task exists, either all tasks have been scheduled, or a circuit of simultaneously dependent nodes is present. At least one task in the circuit must be executed without all of its requisite information to begin the process of iteration. The steps of the algorithm are presented below.

Step 1. Scheduling Independent Tasks:

The first step in this algorithm partitions the design tasks by scheduling tasks which are independent of other tasks. These tasks are easily identified in the matrix as empty rows or columns. The empty row indicates that the task requires no unavailable information and should be scheduled into the earliest available slot. An empty column indicates that the task provides no information to future tasks and should be scheduled into the latest available slot. Once a task is scheduled, it is removed from the matrix and further partitioning analysis is performed on the remaining tasks.

Step 2. Identifying Simultaneously Dependent Tasks

In the design process it is likely that we will encounter tasks which require information that is not available at the present stage of the design. This indicates the existence of a loop of information dependence. When this situation is encountered, the algorithm enumerates the loop using either of two methods described below. Once a loop is found, its members are collapsed into a single composite task and partitioning proceeds. When no tasks remain, the partitioning is complete and the result is a matrix which

is mostly lower triangular except for the blocks along the diagonal. The algorithm is summarized in Table 1.

1. Schedule the tasks which are not components of any loops. Tasks with empty rows have all required information and can be performed first. Tasks with empty columns provide no information required by future tasks and can be performed last. Once a task is scheduled, remove it from further consideration. Repeat until no empty columns or empty rows are found.
2. Identify loops either by path searching or by the powers of the adjacency matrix. Represent all tasks in a loop as a single task.
3. Repeat steps 1 and 2 until all tasks have been scheduled.

Table 1. Partitioning Algorithm

3.1.1 Identifying Loops by Path Searching

The first method to identify loops is referred to in the literature as path searching. Information flow is traced either backwards or forwards until a task is encountered twice [13]. All tasks between the first and second occurrence of the task constitute a loop of information flow. When all loops have been identified, and all tasks have been scheduled, the sequencing is complete and the matrix is in block triangular form. A matrix in this form is mostly lower triangular with blocks of simultaneously dependent tasks placed along the diagonal.

Figure 5 illustrates a simple example of partitioning using path searching for a matrix of seven tasks. The partitioning proceeds as follows: a) The unpartitioned matrix in its original order. b) Task F does not depend on information from any other design tasks as indicated by an empty row. Schedule Task F first in the matrix and remove it from further consideration c) Task E does not provide information to any design tasks in the matrix as indicated by an empty column. Schedule it last in the matrix and remove it from further consideration d) No tasks have empty rows or columns. A loop exists and can be traced starting with any of the remaining tasks. In this case, we select task A and trace its dependence on Task C. Task C is simultaneously dependent upon information from Task A. Since Task A and Task C are in a loop, collapse one into the other and represent them as a single, composite task, Task CA. e) Task CA has an empty column indicating that it is not part of any other loop. Schedule it last and remove it from further consideration. f) Trace dependency starting with any unscheduled task: Task B depends on Task G which depends on

Task D which depends on Task B. This final loop includes all the remaining unscheduled tasks. g) The final partitioned matrix.

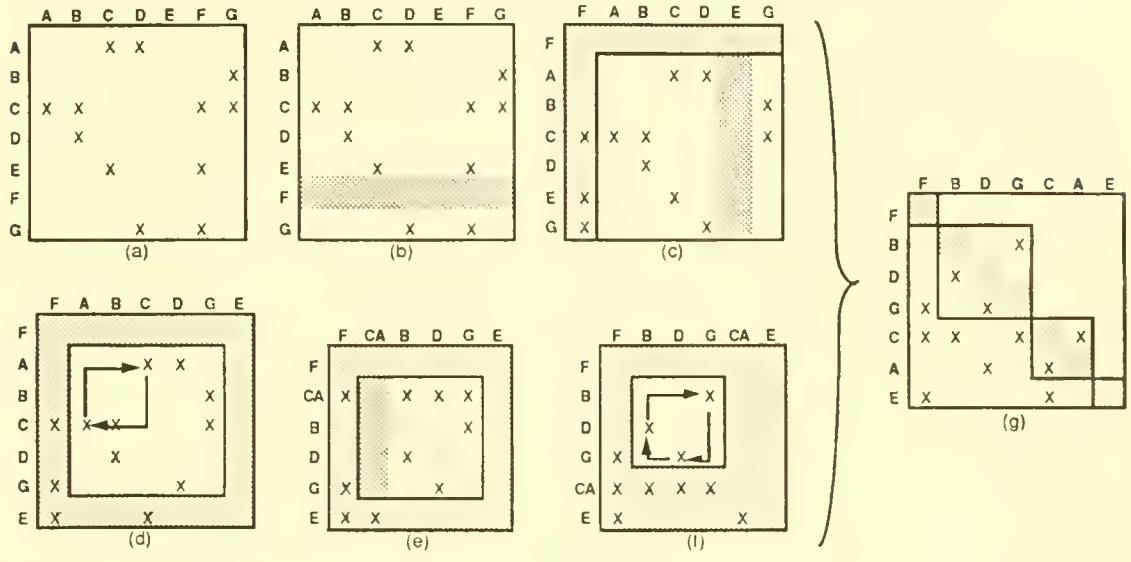


Figure 5. Matrix Partitioning

3.1.2 Identifying Loops by Powers of the Adjacency Matrix

Another method for finding the loops of information flow is a technique which uses powers of the adjacency matrix to identify successively higher order loops [9]. The adjacency matrix is a square matrix identical to the design structure matrix, but with its diagonal elements removed and its non-blank elements replaced with 1's. A non-zero element, a_{ij}, in the adjacency matrix indicates that task i provides direct input (is directly linked) to task j. Raising the binary adjacency matrix to the n-th power using Boolean arithmetic[1] yields a higher-order matrix containing elements a_{ij} indicating that task i can reach task j in n steps.

For example, squaring the adjacency matrix yields a matrix revealing which tasks are reachable in two steps. The cube of the adjacency matrix identifies nodes which are reachable in three steps. The powers of the adjacency matrix are useful for determining the circular flow of information because any task in a circular information flow must be reachable from itself. This implies that a task is reachable from itself in n steps if, in the adjacency matrix raised to the n-th power, the task has a non-zero entry on the diagonal.

---

[1]Boolean arithmetic follows two rules:

a1*a2*a3*... = min(a1,a2,a3,...)

a1+a2+a3+... = max(a1,a2,a3,...)

Figure 6 illustrates use of the adjacency matrices to enumerate iteration loops using the same matrix with seven elements introduced above. The first simplification to make is to notice that Task F does not require information from the other tasks and Task E provides no information to the other tasks. This is sufficient to guarantee that Task E and Task F are not components of any loops and can be removed from the matrix before identifying the circuits. The square of the adjacency matrix reveals that Task A and Task C are in a two step loop. The cube of the adjacency matrix reveals that Tasks B, D, and G are in three step loop. The higher powers of the adjacency matrix reveal no other loops in the system.

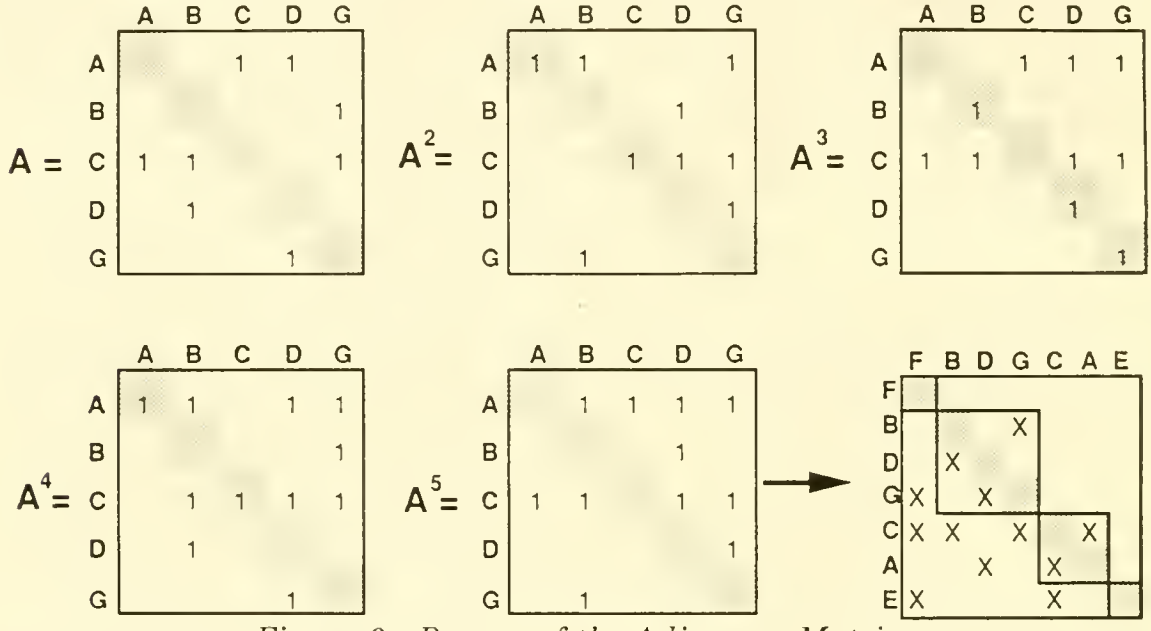


Figure 6. Powers of the Adjacency Matrix

3.2 Tearing Algorithms

Once partitioning has placed the design structure matrix in block-triangular form, the tearing analysis focuses on the ordering of tasks within the blocks. Each loop of information flow revealed by the matrix partitioning can be subjected to analysis using information about the interdependencies. As in partitioning, various approaches can be used. The discussion which follows presents those found most commonly in the literature.

3.2.1 Tearing with Shunt Diagrams

Steward maps the task dependencies of loops into shunt diagrams to identify the most influential tearing points [18]. Shunt diagrams reveal which tears are likely to be most effective in breaking circuits to initialize the iteration. This relies on the manager's knowledge of both the *structure* and the *context* of the proposed tear. The structure indicates that the two

tasks are interdependent; the context is all further information on how the tasks are interrelated. Because this process requires both engineering and management judgement the result of tearing reflects a subjective assessment of which tasks are the least sensitive to incomplete information. The results of tearing will be highly dependent upon the individual's knowledge of the engineering relationship between the tasks.

3.2.2 Tearing by Heuristics

It is possible to order the tasks within a loop using simple heuristics. Rogers uses a knowledge-based software package based on heuristics to minimize the number of feedbacks in any particular loop [11]. The circuits are identified using the path searching technique outlined above, and these loops are analyzed using rule based inferences to decompose the design task into loosely coupled subsystems with minimum pieces of information assumed. The software is flexible enough to accommodate other strategies by varying the rules which are entered.

An algorithm which achieves a minimum amount of information assumed in a circuit has been proposed by Kehat and Shacham [7]. The general strategy is identical to that of Rogers: to identify the task or tasks which are least dependent upon the other tasks and schedule these earliest in the loop. When more than one task requires the same number of inputs, the one which supplies the most information to subsequent tasks is selected first. This basic procedure is applied until all tasks in the loop are scheduled. The details of the algorithm are presented in Table 2.

1. Schedule the tasks which have a minimum number of input streams. (initially there will be none). These are identified in the matrix as tasks with the minimum number of row elements. If there is more than one such task, select the one with the maximum number of output streams. These are identified in the matrix as tasks with the maximum number of column marks. If this fails to identify a unique task, starting with each of the tasks under consideration, compare the number of steps required to reach all the other tasks under consideration. The task which requires the maximum number of steps is selected. If the lists are of equal length, the choice is irrelevant.
2. Remove the scheduled task from consideration and repeat step 1 until all tasks have been scheduled.

Table 2. Tearing Algorithm to minimize the number of tears

It should be noted that various goals can be pursued when performing the tearing analysis. The above criterion to minimize the

number of approximations is just one of many possible objectives. For example, one may try to minimize the number of tasks reliant upon the imperfect information, or minimize the number of task outputs which are estimated. The algorithms for accomplishing these goals using shunt diagrams are discussed in the literature and will not be repeated here [18].

4. Design Process Analysis: Numerical Matrix Approaches

One shortcoming of the representations and techniques presented above is the assumption that all task relations are equal. No attempt is made to differentiate between amounts of information transferred between tasks in the matrix. It is reasonable to expect that certain dependencies will be stronger than others, or that certain transfers of information will be critical. In the design structure matrix, task interactions are not differentiated from one another.

We aim to embody some information about the tasks' interdependencies into the matrix itself and to incorporate this dimension into the analysis techniques applied to the design process. The binary design structure matrix can be vastly improved by having the matrix elements reflect any of a number of different measures of the relation between tasks or the task's relation to the entire design process. These numerical measures can be used most powerfully during tearing analysis.

We have generated new tearing algorithms which take advantage of the increased information available in a numerical design structure matrix [4]. In general, our numerical tearing algorithms are specific to a particular definition of the numerical values in the matrix. One such algorithm we have implemented uses numerical matrix elements representing task repeat probabilities and task durations [15]. The repeat probabilities reflect the likelihood of repeating the task if it proceeds without a particular required input. These values are the off-diagonal elements in the matrix. The diagonal elements contain the durations of the task times as if they were performed independently.

For example, in Figure 7, tasks A and B are tightly coupled; Task A needs the output of Task B and Task B needs the output of Task A. If Task A proceeds without the information from Task B, there is a 20% chance that Task A will have to be repeated once Task B does supply the requisite information. If the other option is chosen and Task B is performed without the input from Task A, the likelihood of repeating Task B, once Task A is completed, will be 40%. This matrix is then used to construct a Markov chain whose value (total expected iteration time) beginning with A or B can be computed using the repeat probabilities and the task durations. This allows each sequence to be scored and compared to other sequences on a common basis. This work and the experimental verification of the matrix predictions of iteration are ongoing and detailed in [15].

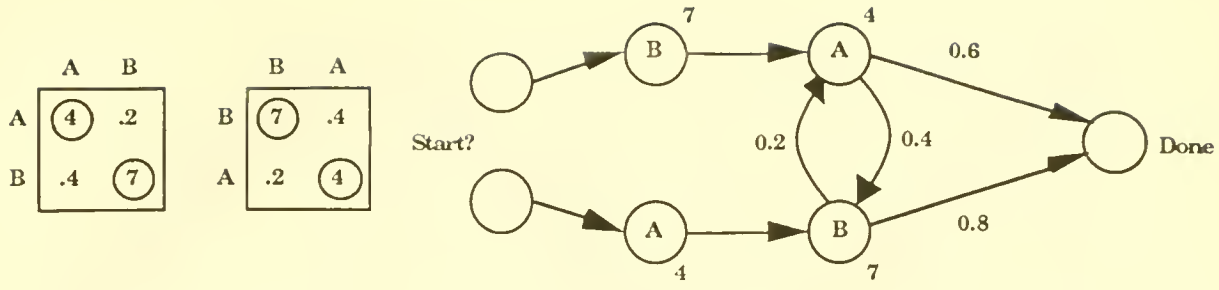


Figure 7. Expected Iteration Computed as the Value of the Markov Chain

We have also implemented several other schemes for scoring sequences of tasks within blocks. One such numerical scoring is based on an estimate of the quantity of information transferred between tasks. A sequence can be scored based on the quantity of information that must be approximated because it is not available when it is required. If a particular ordering requires too much information to be assumed, other sequences can be generated and scored. These scores allow comparisons of different design structures using a common metric.

The numerical algorithm can be aimed at minimizing any number of scores. These scores can be tailored to use a particular metric most appropriately. For example, if one were to measure the certainty with which a particular piece of information could be approximated (perhaps based on previous experience), it would be best to maximize the minimum above diagonal element. These heuristics can easily be changed in the testbed described below to accommodate various measures of task interdependence.

An example of a numerical design structure matrix is presented in Figure 8. In this example, a matrix has been constructed to demonstrate the power of numerical analysis techniques. A twelve by twelve precedence matrix has been converted from binary dependencies to numerical metrics of interdependence. Analysis was performed on the matrix using two methods: (1) standard algorithms which consider all dependencies equal and (2) numerical algorithms which rely upon an enhanced information content.

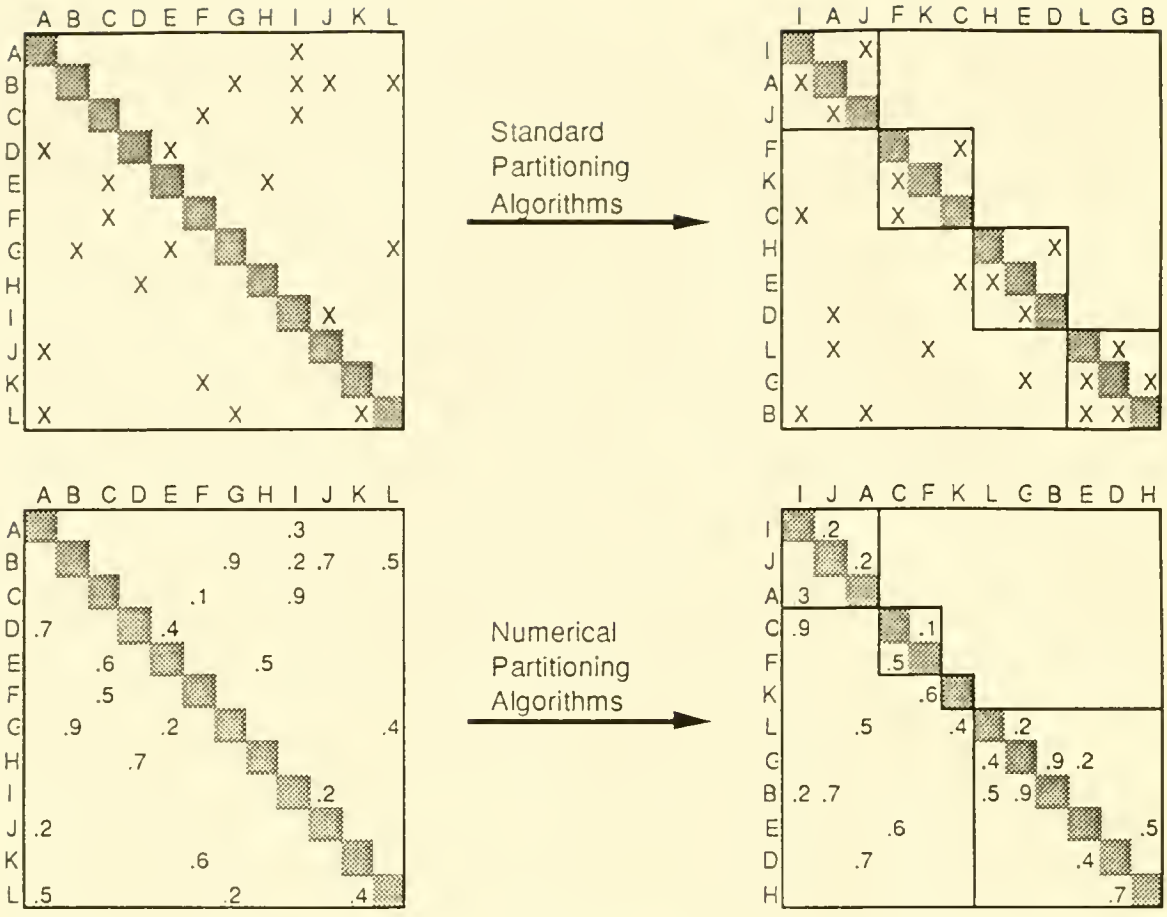

A B C D E F G H I J K L
Standard
Partitioning
Algorithms
I A J F K C H E D L G B
A B C D E F G H I J K L
Numerical
Partitioning
Algorithms
I J A C F K L G B E D H

Figure 8. Numerical Design Structure Matrix

In this example, the goal was to minimize the sum of the above diagonal elements weighted by the distance from the diagonal. Compare the results from standard partitioning to the output from the numerical algorithms. The results are significantly different and lead to different conclusions.

The different results suggest different design management strategies. In the binary matrix, Task G's dependence on Task E would not be considered a strategic one which could impact the execution of the design activity. However, the numerical matrix provides additional insight which could lead to a design organization pictured in Figure 9. Recognizing this tear in the binary matrix would have required much insight or experimentation, while in the numerical matrix, the leverage of this tear is more apparent.

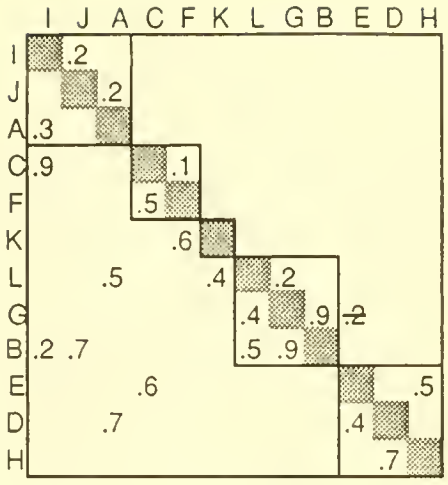


Figure 9. Tear Suggested by Numerical Matrix

The enhanced matrix representation will capture significantly more information in an easily usable form. The enhanced representation capable of handling measures of task interdependence provides enough flexibility to be used in many scenarios and at different levels. A parameter level matrix might measure sensitivity of a parameter to its input. These numerical matrix representations can be analyzed using any number of numerically based tools which utilize the information appropriately to identify both engineering and management improvements.

5. A Testbed for Algorithms

To facilitate experimentation with improved analysis algorithms, we have developed a software platform on which many types of analysis can be implemented and tested. A sample matrix from the software is presented in Figure 10. The software reads a numerical matrix provided by the user and displays it in a simplified format as shown in Figure 10a.

The computer performs the analysis using various algorithms which can be coded and incorporated into the platform. For example, one might choose the matrix elements to represent the variability of the information which is transferred between the tasks. In this case, the algorithm might be coded to minimize the sum of above diagonal elements corresponding to minimizing the uncertainty involved in the design process. Using built-in scoring and swapping functions, the testbed can manipulate the matrix to accomplish goals specified by the user. Interestingly, we have found that coupled with a swapping algorithm, these scoring/swapping schemes can do both partitioning and tearing quite well in one single iterative algorithm.

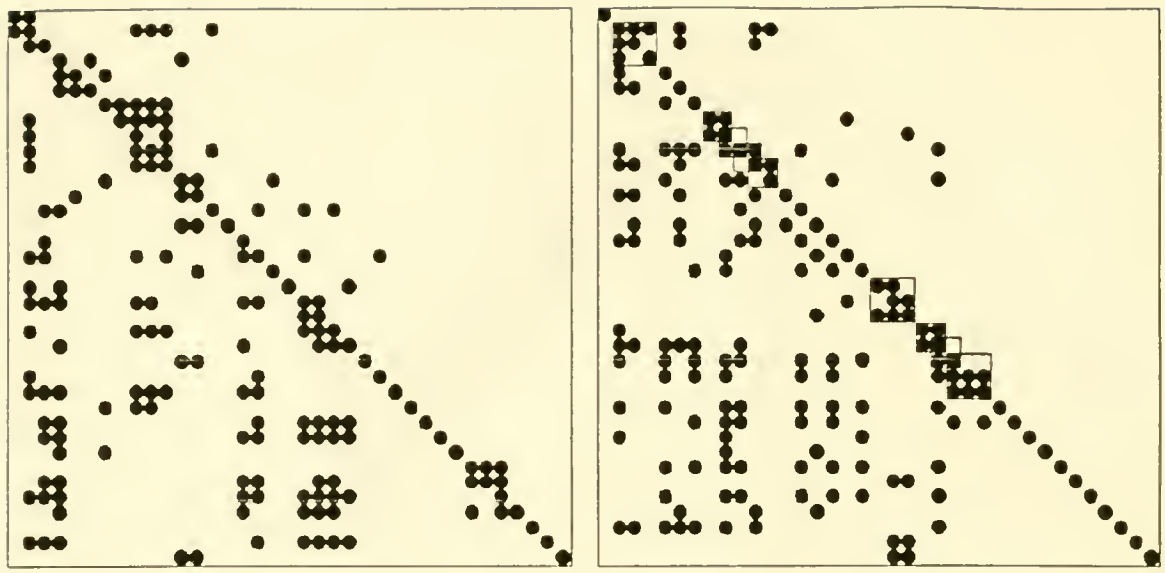


Figure 10. A Typical Display from Testbed Software
Before (a) and after (b) performing automated analysis

In addition, the testbed for matrix analysis can be instructed to provide information about additional features. For example, one may wish to know which tasks, if any, can be performed in parallel with one another. This information is easily obtained by querying the testbed. Another feature presently implemented is the ability the group tasks which are strongly connected into sub-tasks. The sub-tasks are the blocks along the diagonal of the partitioned design structure matrix shown in Figure 10b.

To extend the power of the numerical analysis we hope to have the computer inform the user of specific opportunities available in the output structure. For example, in Figure 8 above, Task G utilizes only 20% of the information transferred to it from Task E. If this information were estimated (the dependency torn), the final block of six elements could be decomposed into two blocks of three tasks each. The additional insight is that this particular tear would allow these two blocks to be performed in parallel. We refer to this as artificial decoupling, and is just one of the strategies presented in [3].

Assessment of a tear requires knowledge of both the engineering interactions and the management requirements. It is not likely that a computer tool will successfully embody these decision capabilities, but enhancing the data provided will clearly allow much richer analysis than is presently available. The effective use of these tools can provide insight into the opportunities available to improve the execution and management of the design process.

6. Conclusion

6.1 Summary of Mappings and Analysis

The matrix representation has the capacity to represent the information requirements of complex design processes. It does not have the practical size limitations of the directed graph and does not suffer from the simplifications required for a PERT chart. A significant advantage of the matrix representation is its ability to concisely represent the complex flow of information. Because the representation is clear and compact, the design structure matrix can provide insight into better management and execution of the design process based on the information flow between the individual tasks.

The existing partitioning and tearing tools can be used to structure the design process in a manner which facilitates the flow of information. If the matrix representation is enriched to capture some quantitative aspect of the design process being modeled, more sophisticated algorithms can provide additional insight into the challenges of design. These tools can be used to identify the best tears based on the enhanced numerical matrix. In addition to identifying tears, these improved analysis tools can provide insight into design management strategies.

6.2 Recommendations for Future Research

The numerical representations and algorithms introduced above rely on our ability to measure or at least estimate some aspect of the design tasks' interdependencies. It would be useful to study a design process with the intent of collecting this kind of information. Once a sufficiently accurate model of this process were determined, it could be analyzed using the tools introduced above. The conclusions reached by analyzing the matrix could be implemented and tested to verify the utility of this design management tool. The first step might be to implement changes in the grouping of tasks suggested by the pattern of information flow.

The techniques presented by Smith [15] could be used to evaluate the design process using different criteria. One particularly useful criteria might be the time required for the design iterations to converge. A restructuring of the design process based on the pattern of information flow might have dramatic impact on the time required to complete the design process. The effects of restructuring the design process may also have an affect on other design variables such as quality or cost, and some of these are being addressed in ongoing work.

Acknowledgements

This research is funded jointly by the National Science Foundation and the MIT Leaders for Manufacturing Program, a partnership involving eleven major US manufacturing firms and MIT's engineering and management schools.

References

[1] C. Alexander. *Notes on the Synthesis of Form.* Harvard University Press, Cambridge, Massachusetts, 1964.

[2] T. A. Black. *A Systems Design Methodology Applied to Automotive Brake Design*, S.M. Thesis, MIT, 1990.

[3] S. D. Eppinger. *Model-Based Approaches to Managing Concurrent Engineering.* International Conference on Engineering Design, Zurich, August, 1991.

[4] S. D. Eppinger, D. E. Whitney, R. P. Smith and D. A. Gebala. *Organizing the Tasks in Complex Design Projects.* ASME Design Theory and Methodology Conference, Chicago, IL, 1990.

[5] F. Harary. *Graph Theory.* Addison-Wesley, Reading, Mass., 1969.

[6] D. M. Himmelblau. "Decomposition of Large Scale Systems, Part 1: Systems Composed of Lumped Parameter Elements", *Chemical Engineering Science.* vol. 21, pp. 425-438, 1966.

[7] E. Kehat and M. Shacham. "Chemical Process Simulation Programs, Part 2: Partitioning and Tearing of System Flowsheets", *Process Technology International.* vol. 18, no. 3, pp. 115-118, March, 1973.

[8] G. Kron. *'Diakoptics', Piecewise Solution of Large Scale Systems of Equations*, Ph.D. Thesis, University of Texas, Austin, 1963.

[9] W. P. Ledet and D. M. Himmelblau. "Decomposition Procedures for the Solving of Large Scale Systems", *Advances in Chemical Engineering.* vol. 8, pp. 185-254, 1970.

[10] D. A. Marca and C. L. McGowan. *SADT: Structured Analysis and Design Technique.* McGraw Hill, New York, 1988.

[11] J. L. Rogers. *DeMAID: A Design Manager's Aide for Intelligent Decomposition User's Guide.* NASA Technical Memorandum 101575, March, 1989.

[12] D. T. Ross. "Structured Analysis (SA): A Language for Communicating Ideas", *IEEE Transactions on Software Engineering.* vol. SE-3, no. 1, pp. 16-34, January, 1977.

[13] R. W. H. Sargent and A. W. Westerberg. "'Speed-Up' in Chemical Engineering Design", *Trans. Instn. Chem. Engrs.* vol. 42, T190-T197, 1964.

[14] H. A. Simon. *The Sciences of the Artificial.* MIT Press, Cambridge, Massachusetts, 1970.

[15] R. P. Smith and S. D. Eppinger. "Modeling Design Iteration", *Working Paper, MIT Sloan School of Management.* no. 3160-90, June, 1990.

[16] J. Sobieszczanski-Sobieski. *Multidisciplinary Optimization for Engineering Systems: Achievements and Potential.* NASA Technical Memorandum 101566, March, 1989.

[17] D. V. Steward. "The Design Structure System: A Method for Managing the Design of Complex Systems", *IEEE Transactions on Engineering Management.* pp. 71-74, August, 1981.

[18] D. V. Steward. *Systems Analysis and Management: Structure, Strategy, and Design.* Petrocelli Books, New York, 1981.

[19] J. N. Warfield. "Binary Matrices in System Modeling", *IEEE Transactions on Systems, Man, and Cybernetics.* vol. SMC-3, no. 5, pp. 441-449, September, 1973.

[20] J. D. Wiest and F. K. Levy. *A Management Guide to PERT/CPM.* Prentice-Hall, Englewood Cliffs, New Jersey, 2nd Edition, 1977.

6809 078

Date Due

DEC 2 2 2003

OCT. 28

FEB. 08 1994

MAY 09

JAN. 09 1995

Lib-26-67

MIT LIBRARIES DUPL

3 9080 00686242 6